Simulation of sheared suspensions with a parallel implementation of QDPD

James S. Sims and Nicos Martys
National Institute of Standards and Technology
Gaithersburg, MD 20899-8911

Correspondence to:
James S. Sims
National Institute of Standards and Technology
100 Bureau Drive Stop 8911
Gaithersburg, MD 20899-8911
jim.sims@nist.gov
phone - 301-975-2710
fax - 301-975-3218

PACS numbers: 83.80.H, 83.10.R, 83.85.P

keywords: Dissipative particle dynamics, mesoscopic modeling, parallel algorithms, rheology, spatial decomposition, suspensions

Abstract

A parallel quaternion-based dissipative particle dynamics (QDPD) program has been developed to study the flow properties of complex fluids subject to shear. The parallelization allows for simulations of greater size and complexity and is accomplished with a parallel link-cell spatial decomposition using MPI[1]. Rigid body inclusions spread out across processors, the algorithm implications of the DPD random force, and the sheared boundary condition are all discussed. Results are presented for two different distributed memory architectures. Parallel efficiencies were significatly improved for benchmark calculations.

I. Introduction

The flow properties of suspensions (e.g., colloids, ceramic slurries, and concrete) is of fundamental interest and important for many technological applications. While some analytical solutions describing the rheological properties of simple suspensions exist, e.g. very dilute suspensions, understanding the flow of more complex suspensions (e.g. dense suspensions, suspensions composed of particles that interact, etc.) remains a great challenge. Modeling such systems by computational fluid dynamics, while promising, remains a challenge because it is difficult to track boundaries between different fluid and solid phases. Recently, a new computational method called dissipative particle dynamics (DPD) [2] has been developed which holds promise for modeling such complex fluids. Indeed DPD may have some advantages over other computational fluid dynamics methods because DPD can naturally accommodate many boundary conditions while not requiring meshing (or remeshing) of the computational domain.

On the surface, DPD looks very much like a molecular dynamics algorithm [3] (MD) where particles, subject to interatomic forces, move according to Newton's laws. However, the particles in DPD are not atomistic, but instead are a mesoscopic representation of the fluid. As a result, the interactions between the DPD particles are not directly based on a Lennard-Jones potential, but are typically produced by three classes of forces: conservative forces, dissipative forces, and a random force. The conservative force is simply a central force, derivable from some potential. The dissipative force is proportional to the difference in velocity between particles and acts to slow down their relative motion. The dissipative force can be shown to produce a viscous effect. The random force (usually based on a Gaussian random noise) helps maintain the temperature of the system while producing some viscous effects. It can be shown that, in order to maintain a well defined temperature by way of consistency with a fluctuation-dissipation theorem [4], coefficients describing the strength of the dissipative and random forces must be coupled. By mapping of the DPD equations of motion to the Fokker-Planck equation [5], it has been demonstrated that the DPD equations can recover hydrodynamic behavior consistent with the Navier-Stokes equations.

To model a rigid body inclusion in a fluid, a subset of the DPD particles are initially assigned a location in space such that they approximate the shape of the object [6]. The motion of these particles is then constrained so that their relative positions never change. The total force and torque are determined from the DPD particle interactions and the rigid body moves according to the Euler equations. For our simulations we use a quaternion-based scheme developed by Omelayan [7] and modified by Martys and Mountain [8] for a velocity-Verlet algorithm to integrate the equations of motion. Finally, we use a Lees-Edwards boundary condition [3, pp 246-247] to produce a shearing effect akin to an applied strain at the boundaries.

Because of the complexity and size of the simulations (modeling a suspension of tens of thousands of different shaped rigid bodies using millions of DPD particles), it is desirable to utilize a parallel computation scheme to make such calculations tractable over a reasonable period of time.

Two types of parallelization approaches are usually considered for MD codes. The first is rather straightforward and involves splitting up the calculation of forces between particles over the processors. We refer to this as a "replicated data" parallelization scheme. However, as is well known, such an approach, for parallelization of codes, can involve a large amount of message passing and as a result the performance of the code may quickly degrade with increasing number of processors. An alternative approach is to spatially decompose the computational domain between the processors. For molecular dynamics simulations the latter approach (spatial decomposition) has been shown to significantly reduce the amount of message passing, allowing for better scaling of the number of computational cycles performed (for a fixed period) with increasing number of processors.

In the process of developing a spatial decomposition version of our DPD code it became apparent that our algorithms include features, somewhat different than typical molecular dynamics codes, that need to be properly handled in the parallelization. First, contributions from random forces between DPD particles may need to be passed between processors so that momentum is conserved. Second, as we are utilizing a Lees-Edwards boundary condition (of course this feature is not unique to DPD), the DPD particles, rigid bodies and related information may need to be passed to processors that are not necessarily neighbors in the sense of the usual periodic boundary conditions. Third, as the rigid bodies come in a variety of shapes and sizes that may span a region bigger than the computational domain of several processors, we need to properly account for all the forces contributing to their motion. In this paper we describe our algorithms which extend the usual spatial decomposition approach to account for these features in our DPD code. We also show some results concerning the performance of the our code on several different computational platforms.

II. QDPD Spatial Decomposition

A. Outline of the parallel algorithm

QDPD was originally written in Fortran 77 as a sequential program. As with most molecular dynamics alogrithms, a signifi cant fraction of the computation time involves calculating forces between particles. In DPD the forces are short-range, decaying rapidly with separation, which means that only nearby particles, closer than some cutoff distance r_c, need be considered. We use an implementation of the link-cell method of Quentrec and Brot [9], described in Allen and Tildesley's book [3, pp 149-152], to construct neighbor lists.

Since the sequential link-cell algorithm breaks the simulation space into domains, it seems natural to map this geometrical, or domain, decomposition onto separate processors. Doing so is the essence of the parallel link-cell technique [10, 11] [1]. By subdividing the physical volume among processors, most of the computation becomes local and communication is minimized. There is, in principle, a 1/ P scaling, where P is the number of processors, with a correction due to message passing between processors. Hence, the spatial decomposition approach may be effective for distributed-memory computers and networks of workstations. Our parallelization of this algorithm was accomplished with the Message Passing Interface (MPI) library [1].

The basic idea of our spatial decomposition follows. Split the total volume into P domains. If we choose a 1D decomposition ("slices of bread"), then the pth processor is responsible for particles whose x-coordinates lie in the range

$$(p-1)L_x/P \leq x < pL_x/P. \tag{1}$$

Similar equations apply for 2D and 3D decompositions for simulation dimensions L_y and L_z. An algorithm due to Plimpton [13] is used to assign *P* processors to a 3D box so as to minimize the surface area (and hence, yield a good load balancing).

Each processor runs a link-cell program corresponding to a particular domain of the simulation box. For example[2], in Figure 1 nine processors were used to

[1]See Plimpton [12] for excellent discussions of all fast parallel algorithms.

[2]Simulations are in 3D but illustrations are in 2D.

divide a 2D simulation space into domains, each processor being assigned to one of the nine domains[3]. Each domain is then divided into cells for the link-cell algorithm. To complete the force calculation on particles in cells at the interface between processor domains, each processor needs to know information about the particles in the adjacent cells, which are found on a neighboring processor. At each timestep we communicate this information across the interface between adjacent processors in these edge cells. The information that has to be passed are the particle positions and velocities needed for the forces calculations and a unique particle label. The unique particle label is needed because, in DPD, different random forces are associated with each particle pair. As forces are considered between two particles i and j, the particle identifi ers are used to communicate this force back to the processor which "owns" j (the "pair" particle of i) and add it (with the appropriate sign) to the forces acting on particle j[4].

To understand the communication between processors, consider processors 3,4,5 in Fig 1. Dashed lines are used to show the cells in processors 3 and 5 which are adjacent to 4. Information about the particles in these dashed line cells is communicated to 4, making up "ghost" cells on 4. To complete the "extended volume" needed on processor 4 to compute the forces on all the particles it owns, information is communicated (swapped) across the interface between adjacent processors in the y direction as well. To account for the cross-hatched corner cells, the swap in y includes information about not only particles that the processor owns but also information about "ghost" particles in ghost cells. So processor 7 sends information about particles in the dashed line cells as well as the cross-hatched cells (obtained from processors 6 and 8) to processor 4. At this point processor 4 has all the information it needs to calculate forces on all the particles it owns, and similarly all of the other processors have all the information they need. These exchanges of data can be achieved by one set of communications between the processors. A processor only has to communicate once with all of its neighbors, so each processor communicates with at most four other processors (six in 3D), rather than, say 64 in a 64 processor replicated data calculation.

Once the extended volume has been built, a link-cell list of all particles in the original volume plus the extended volume can be built. Looping over the particles in the original volume calculates the forces on them and their pair particles. An extra set of communications between processors (the reverse of the commu-

[3]Each processor is assigned an index, where indices start at 0.

[4]Since, by Newton's third law, the force of i on j must be equal in magnitude and opposite in sign to the force of j on i.

nication swaps setting up the ghost cells) is done at this point to send pair particle forces back to the processors which own the pair particles in the extended volume. Now we can calculate the new positions of all particles and move the particles which have left a processor to their new "home" processor. If particles move into domains controlled by other processors, information about the particle must be moved to its new home processor. Again these exchanges of data can be achieved by one set of communications between the processors. In this set of communications, *all* information about a particle needed for one time step must be communicated, not just the information needed for the forces calculation.

B. Further details

The above description summarizes many features of the spatial decomposition approach, with the following additions. First, following Plimpton [13], we distinguish between owned particles and ghost particles, those particles that are on neighboring processors and are part of the extended volume on any given processor. For ghost particles, only the information needed to calculate forces is communicated to neighboring processors. Second, for suspensions, there are two types of particles, free "fluid" particles and particles belonging to rigid bodies. A novel feature of this work is that we explicitly do *not* keep all particles belonging to the same rigid body on the same processor. Since the largest rigid body that might be built can consist of a significant fraction of all DPD particles, it would be difficult if not impossible to handle such objects without serious load-balancing implications. Each processor contains data sets for all rigid bodies consisting of lists of particles with a unique object label. Every processor computes rigid body properties (i.e. total forces, torques) contributed by each particle it owns, and these properties are globally summed so that all processors have the same solid inclusion properties. Since there are only a small number of rigid bodies (relative to the number of particles), the amount of communication necessary for the global sums is small and the amount of extra memory is also relatively small. Hence it is an effective technique.

The formation of extended volumes using ghost cells requires additional explanation. To accomodate the ghost cells, the number of cells in each direction is increased by 2. So, for example, for a division of the central processor into 100 cells as in Figure 1, the number of cells in the x and y directions is 10. To accomodate the left and right ghost cells, the x and y cell dimensions, M_x and M_y

respectively, (including ghost cells) becomes 12 (10 + 2). We use the following to defi ne a cell index for particle i (ICELL$_i$)

$$ICELL_i = I_{x_i} + (I_{y_i} - 1)M_x + (I_{z_i} - 1)M_xM_y, \tag{2}$$

where I_{x_i}, I_{y_i}, and I_{z_i} are now given by

$$I_\alpha = 1 + INT((r_{\alpha_i}S_\alpha + 0.5)M_\alpha), \tag{3}$$

with r_{α_i} the coordinate of particle i, $\alpha = x, y$ or z, and S_α are scale factors whose purpose is to transform coordinates so that a processor's owned particles in a domain will have values in the range

$$2 \leq I_\alpha \leq M_\alpha - 1. \tag{4}$$

In Figure 2 we show the central processor from Figure 1 again, with its own and ghost cells renumbered according to the above. Using these scale factors, it is straightforward to identify which particles need to be passed in all 4 (or 6) directions. For example, particles whose I_{x_i} value is 2 are left edge particles and need to be passed to the processor to the left; particles whose I_{x_i} value is 11 (M_x - 1) are right edge particles and need to be passed to the processor on the right. It is important to note that particles in ghost cells are included in subsequent swaps, so, for example, particles whose I_{y_i} value is 2 are passed down (including particles in the ghost cells with I_{y_i} = 1 and 12), and particles whose I_{y_i} value is 1 are passed up. In this the way the particles in corner cells are made available to adjacent processors. As processor 4 communicates information about particles in its edge cells with I_x = 11 to processor 5, processor 5 in turn communicates information about particles in its left edge cells to processor 4, which become the right edge ghost cells on processor 4. So after swapping with processors to its left, right, north, and south, the complete "extended volume" exists on processor 4, and this can be followed by the link-cell list construction (I_x = {1,12}, I_y = {1,12}) and computation of forces (for particles owned by this processor, which are those in cells with I_x = {2,11}, I_y = {2,11}).

Now consider Figure 1 again, and imagine calculating the forces using a single processor and the link-cell algorithm, and subdividing the simulation box into 30 cells in x and y. The force calculation on particles in cells with I_x, I_y = {11,20} in Figure 1 would be calculated exactly the same way as the particles owned by processor 4 in Figure 2, for which I_x, I_y = {2,11}, which is the essence of the parallel link-cell method.

An important point that was skipped in the above discussion is the treatment of the shear boundary conditions. In Figure 3 we show the Figure 1 simulation box again, and three boxes above the simulation box, moving to the right, as well as three boxes below the simulation box, moving to the left. $0'$, $1'$, and $2'$ are images of 0, 1, and 2 which have moved to the right because of the shear. $6'$, $7'$, and $8'$ have moved left. We also show the sheared upper boundary and the extended volume we have to build prior to computing forces. Because of Newton's third law, the extended volume we need includes left, right, and up layers, but not down ($I_y = 1$). Care must be taken to include the shear shown in the figure. This is accomplished by forming the y ghost layer before x (for 3D, the order is z, y, x). The $I_y = 32$ layer is formed by processors 0, 1, and 2 sending their $I_y = 2$ cells to processors 6, 7, and 8 respectively, and adding the simulator box distance in y. Consider the $I_y = 2$ layer from processor 0 which now forms a ghost layer on top of 6. The particles in this top ghost layer are now shifted to the right according to the Lees-Edwards boundary condition [3, pp 149-152]. As a result of this translation, some of the ghost particles on 6 will, in this case, need to be sent to processor 7 (the shifted ghost layer originally totally on top of 6 is now the I_y = 32 ghost layer bounded by the edges of the $0'$ cell). When this is done, it is important to keep track of the communications so they can be reversed in the transfer of Newton's third law forces back to the home processor of the ghost particles. The Lees-Edwards boundary condition is now satisfied with these maneuvers, that is, particles leaving at the bottom of the simulation box enter at the top ghost layer (the mirror image) with their x coordinate shifted to account for the strain.

III. Results and Discussion

Figure 4[5], shows the performance of our codes on two distributed memory architectures. For the replicated data version of our code, the best we could do was a factor of 4.3 improvment on 16 processors on a Linux cluster with Myrinet. In comparison, the spatial decomposition version of the code, running on the same Linux cluster showed a greatly enhanced performance (a factor of 10.5 on 16 processors). The best results, for the spatial decomposition version, show a speed up of a factor of 24 on 27 200MHz Power3 processors on an IBM SP2 [6] , a distributed

[5] In the figure we plot normalized processing time = (time to complete benchmark run on multiple processors) / (time to compute benchmark run on a single processor).

[6] The identification of any commercial product or trade name does not imply endorsement or recommendation by the National Institute of Standards and Technology.

memory cluster, but with a high-speed interconnect which allows it to approach the scalability of a shared memory machine in many cases.

Not surprisingly, our experience is that shared memory machines perform best for this type of algorithm. For example, our spatial decomposition code has proven effective in a shared memory environment [14] as well, where the speedups are a factor of 29 on 32 processors of an SGI Origin 3000 system and a factor of 50 on 64 processors of the same system. In contrast, for the replicated data parallelization, speedups of a factor of 17.5 on 24 processors of an SGI Origin 3000 [14]. Clearly, communication costs quickly become prohibitive for replicated data parallelizations on distributed memory architectures. Scaling to a very large number of processors is poor even in the shared memory environment, and it makes the replicated data approach almost unusable on distributed memory machines including those with high-speed interconnects like the IBM SP2 cluster.

IV. Summary

In adopting a spatial decomposition approach, we found a signifi cant improvment in performance of our codes despite the additional complications of communicating the random forces, implimentation of the Lees-Edwards boundary condition, and accounting for objects that can extend over many processor domains. Clearly, the main bottleneck of such an approach is the message passing between processors. As such technologies improve, we expect corresponding improvements in the computional performance of our algorithms.

Acknowledgements

We would like to thank John G. Hagedorn for useful comments and programming support, Robert Bohn and N. Alan Heckert for graphics support, and Chris Schanzle for systems support.

References

[1] Message Passing Interface Forum, Int. J. Supercomput. Appl. **8** (3/4), 159 (1994).

[2] P. J. Hoogerbrugge and J. M. V. A. Koelman, Europhys. Lett. **19** (1), 155 (1992).

[3] M. P. Allen and D. J. Tildesley, Computer simulation of liquids, Clarendon Press, Oxford (1987).

[4] P. Español and P. Warren, Europhys. Lett **30**, 191 (1995).

[5] C. Marsh, G. Backx, and M. H. Ernst, Europhys. Lett **38**, 441 (1997).

[6] J. M. V. A. Koelman and P. J. Hoogerbrugge, Europhys. Lett. **21** (1), 363 (1993).

[7] I. Omelyan, Computer Physics **12**, 97 (1998).

[8] N. S. Martys and R. D. Mountain, Phys. Rev. E **59** (3), 3733 (1999).

[9] B. Quentrec and C. Brot, J. of Comput. Phys. **13**, 430 (1973).

[10] M. Pinches, D. Tildesley and W. Smith, Mol. Simul. **6**, 51 (1991).

[11] W. Smith, Comput. Phys. Commun. **67**, 392 (1992).

[12] S. J. Plimpton, J. of Comput. Phys. **117**, 1 (1995).

[13] S. J. Plimpton, R. Pollock, and M. Stevens, Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, March 1997.

[14] J. S. Sims, J. G. Hagedorn, P. M. Ketcham, S. G. Satterfield, T. J. Griffin, W. L. George, H. A. Fowler, B. A. am Ende, H. K. Hung, R. B. Bohn, J. E. Koontz, N. S. Martys, C. E. Bouldin, J. A. Warren, D. L. Feder, C. W. Clark, B. J. Filla, and J. E. Devaney, J. Res. Natl. Inst. Stand. Technol. **105**, 875 (2000).

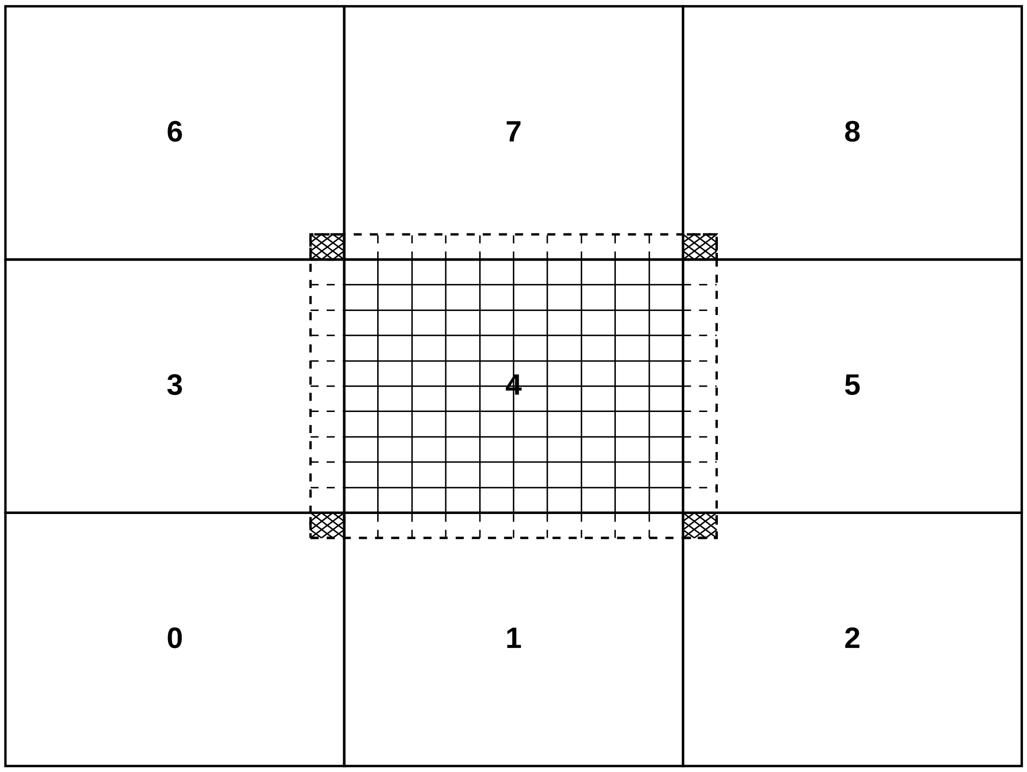

Fig. 1. A nine processor 2-D domain. The small rectangles are cells associated with the link-cell algorithm. The dashed lines correspond to the ghost cells.

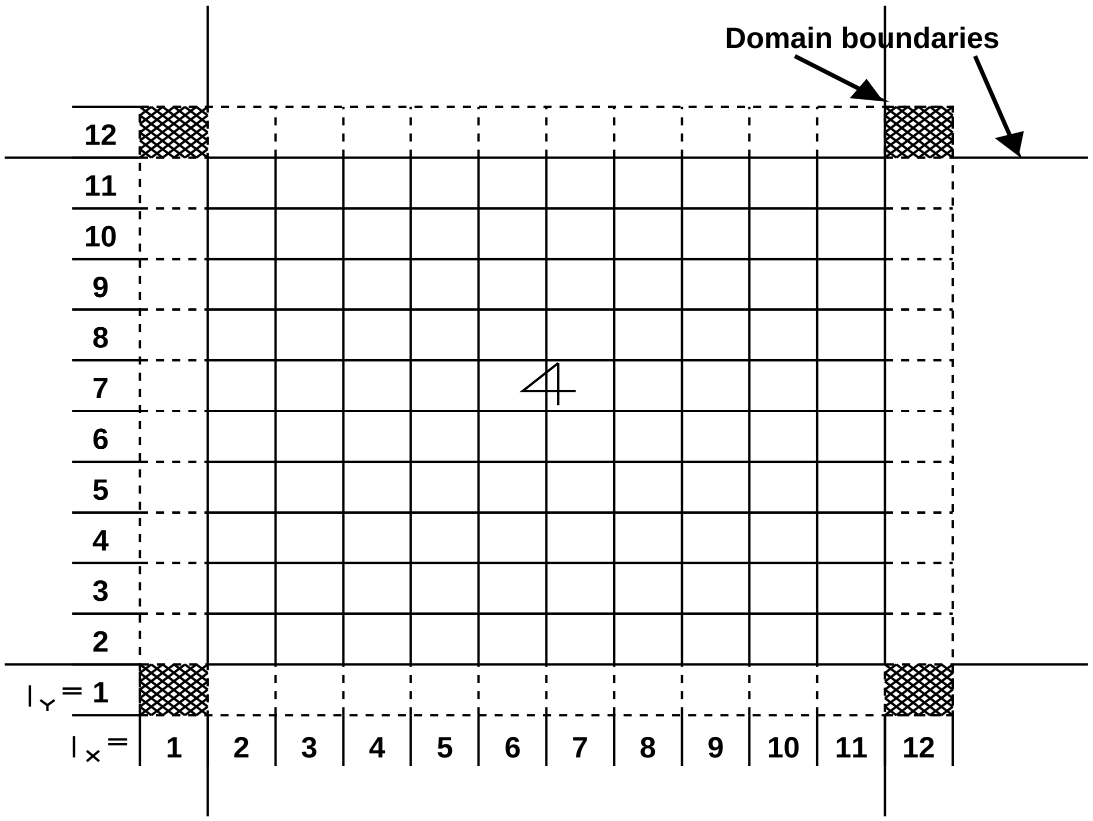

Fig. 2. Enlargement of the central region of Figure 1. Link-cell boundaries become processor domain boundaries. Dashed lines correspond to ghost cells.

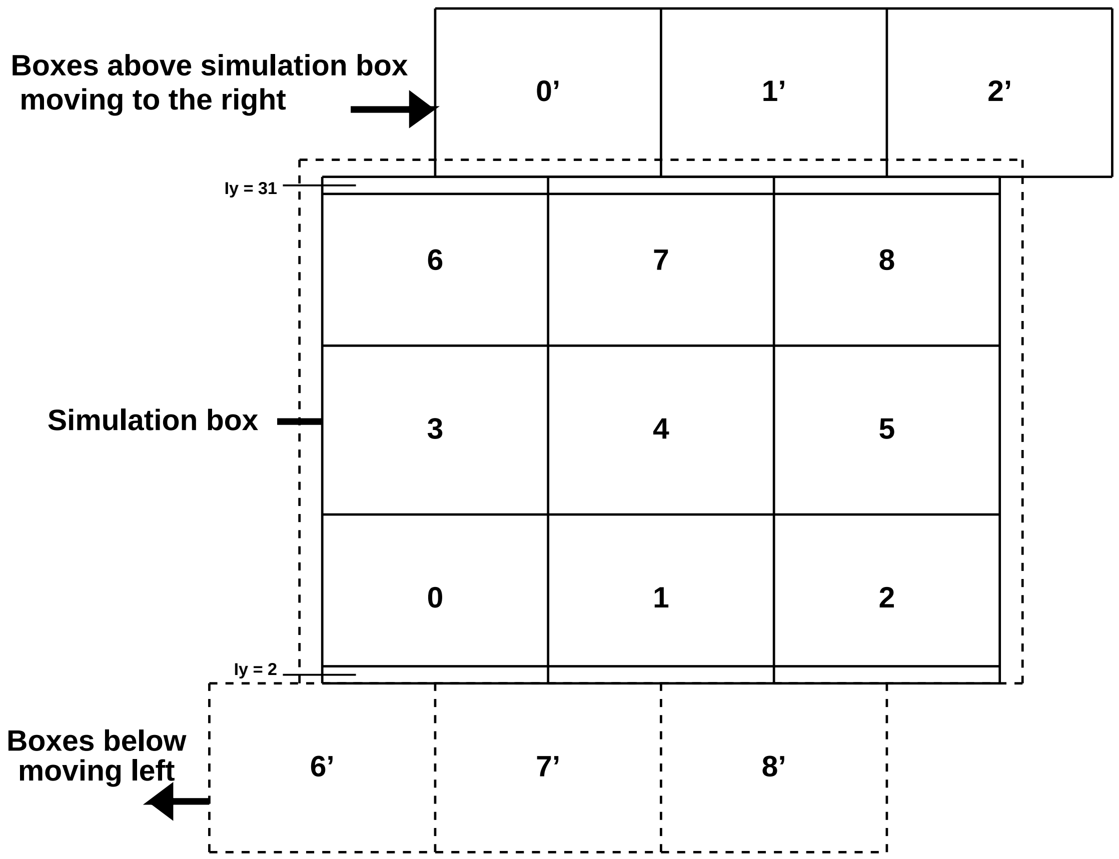

Fig. 3. A nine processor 2-D domain decomposition and neighboring layers resulting from application of an applied strain consistent with the Lees-Edwards boundary condition.

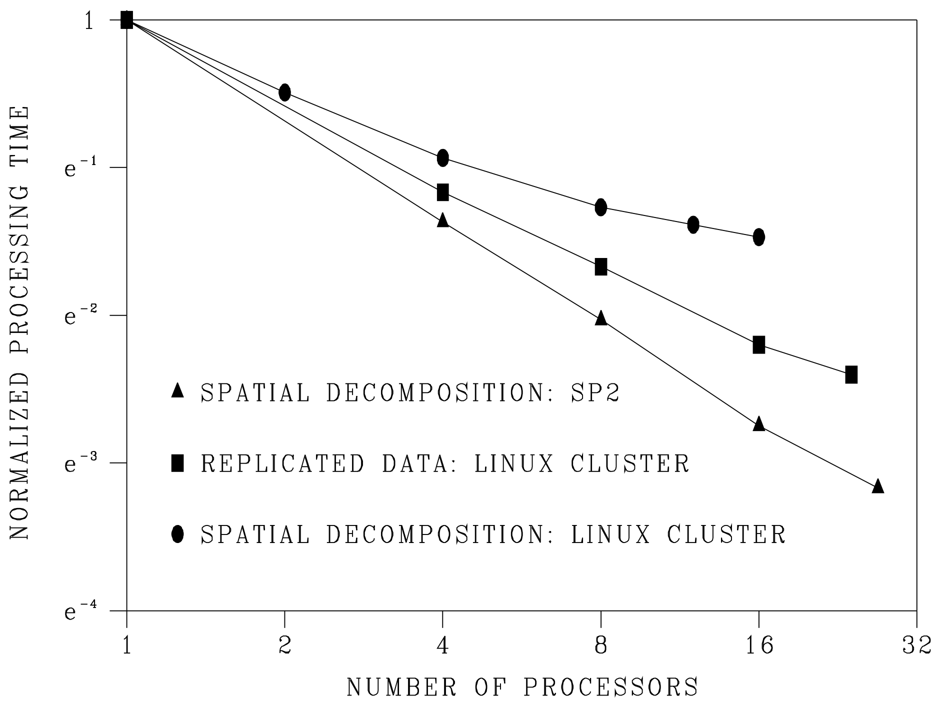

Fig. 4. Logarithm (base e) of normalized CPU time (seconds) versus number of processors. The performance of the replicated data version degrades much more quickly than the spatial decomposition version of the same code.